HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA: CNTT CLC

MÔN: LẬP TRÌNH PYTHON

BÁO CÁO BÀI TẬP LỚN SỐ 2

Giảng viên hướng dẫn	**: KIM NGỌC BÁCH**
Họ và tên sinh viên	**: NGUYỄN CẢNH ĐỨC ANH**
Mã sinh viên	**: B23DCCE006**
Lớp	**: D23CQCE06-B (CLC)**
Ngày sinh	**: 14-10-2005**
Nhóm	**: CÁ NHÂN**

Hà Nội – 06/06/2025

1. Giới thiệu

Bài báo cáo này trình bày quá trình xây dựng và huấn luyện một mô hình mạng nơ-ron tích chập (CNN) để thực hiện bài toán phân loại hình ảnh trên bộ dữ liệu CIFAR-10. Mục tiêu là xây dựng một mô hình có khả năng phân loại chính xác 10 lớp đối tượng khác nhau trong bộ dữ liệu.

Bộ dữ liệu CIFAR-10 bao gồm 60,000 ảnh màu kích thước 32x32 pixel, được chia thành 10 lớp, mỗi lớp có 6,000 ảnh. Có 50,000 ảnh cho tập huấn luyện và 10,000 ảnh cho tập kiểm thử.

2. Xây dựng Mô hình CNN

Xây dựng một mạng CNN với 3 lớp tích chập (Convolutional Layer), theo sau là các lớp pooling và các lớp kết nối đầy đủ (Fully Connected Layer) để thực hiện phân loại.

*** Kiến trúc mô hình:**

1. **Lớp Tích chập 1 (Conv1):**
    - Input channels: 3 (ảnh màu RGB)
    - Output channels: 32
    - Kernel size: 3x3
    - Activation: ReLU
    - Max Pooling: 2x2
2. **Lớp Tích chập 2 (Conv2):**
    - Input channels: 32
    - Output channels: 64
    - Kernel size: 3x3
    - Activation: ReLU
    - Max Pooling: 2x2
3. **Lớp Tích chập 3 (Conv3):**
    - Input channels: 64
    - Output channels: 128
    - Kernel size: 3x3
    - Activation: ReLU

- Max Pooling: 2x2

4. **Các lớp kết nối đầy đủ (Fully Connected):**
    - Làm phẳng (Flatten) output từ lớp tích chập cuối cùng.
    - **FC1:** Input features (tính toán dựa trên output của lớp Conv3), Output features: 128, Activation: ReLU.
    - **FC2:** Input features: 128, Output features: 10 (tương ứng 10 lớp của CIFAR-10).

5. Mô tả chi tiết

- **Nguồn gốc**: Canadian Institute For Advanced Research (CIFAR). https://docs.ultralytics.com/vi/datasets/classify/cifar10/ *(Ấn chuột phải => Chọn "Open Hyper Link")*
- **Nội dung**: Tập dữ liệu bao gồm 60.000 hình ảnh màu. https://docs.ultralytics.com/vi/datasets/classify/cifar10/ *(Ấn chuột phải => Chọn "Open Hyper Link")*
- **Kích thước hình ảnh**: Mỗi hình ảnh có kích thước 32x32 pixel.
- **Số lượng lớp**: Có 10 lớp hoàn toàn loại trừ lẫn nhau. Ví dụ, lớp "ô tô" bao gồm xe sedan và SUV nhưng không bao gồm xe bán tải, trong khi lớp "xe tải" chỉ đề cập đến các xe tải lớn và cũng loại trừ xe bán tải.

Bảng 1: Chi tiết Kiến trúc CNN

Tên Lớp	Loại Lớp	Kênh Đầu Vào	Kênh Đầu Ra	Kích thước Kernel	Bước nhảy (Stride)	Đệm (Padding)	Hàm Kích Hoạt	Kích thước Đầu Ra (batch_size, C, H, W)
Conv1	nn.Conv2d	3	32	3x3	1	1	-	[bs, 32, 32, 32]
ReLU1	nn.ReLU	32	32	-	-	-	ReLU	[bs, 32, 32, 32]
Pool1	nn.MaxPool2d	32	32	2x2	2	0	-	[bs, 32, 16, 16]
Conv2	nn.Conv2d	32	64	3x3	1	1	-	[bs, 64, 16, 16]

ReLU2	**nn.ReLU**	**64**	**64**	**-**	**-**	**-**	**ReLU**	**[bs, 64, 16, 16]**
Pool2	**nn.MaxPool2d**	**64**	**64**	**2x2**	**2**	**0**	**-**	**[bs, 64, 8, 8]**
Conv3	**nn.Conv2d**	**64**	**128**	**3x3**	**1**	**1**	**-**	**[bs, 128, 8, 8]**
ReLU3	**nn.ReLU**	**128**	**128**	**-**	**-**	**-**	**ReLU**	**[bs, 128, 8, 8]**
Pool3	**nn.MaxPool2d**	**128**	**128**	**2x2**	**2**	**0**	**-**	**[bs, 128, 4, 4]**
Flatten	**-**	**128**	**2048**	**-**	**-**	**-**	**-**	**[bs, 2048]**
FC1	**nn.Linear**	**2048**	**128**	**-**	**-**	**-**	**-**	**[bs, 128]**
ReLU_FC1	**nn.ReLU**	**128**	**128**	**-**	**-**	**-**	**ReLU**	**[bs, 128]**
FC2	**nn.Linear**	**128**	**10**	**-**	**-**	**-**	**-**	**[bs, 10]**

3. Huấn luyện, Xác thực và Kiểm thử

Quy trình thực hiện bao gồm các bước sau:

- Tải và Chuẩn bị dữ liệu: Tải bộ dữ liệu CIFAR-10 bằng torchvision.datasets. Áp dụng các phép biến đổi (transform) như chuyển đổi sang Tensor và chuẩn hóa (normalize) giá trị pixel. Dữ liệu được chia thành tập huấn luyện (training), tập xác thực (validation) và tập kiểm thử (testing).
- Định nghĩa Hàm mất mát và Trình tối ưu hóa:
    - Hàm mất mát (Loss Function): Sử dụng CrossEntropyLoss vì đây là bài toán phân loại đa lớp.
    - Trình tối ưu hóa (Optimizer): Sử dụng Adam để cập nhật trọng số của mạng.
- Vòng lặp Huấn luyện:
    - Huấn luyện mô hình qua một số lượng epoch nhất định.
    - Trong mỗi epoch, mô hình sẽ học trên tập huấn luyện và được đánh giá trên tập xác thực để theo dõi hiệu suất và tránh overfitting.
    - Lưu lại các giá trị loss và accuracy của cả tập huấn luyện và tập xác thực sau mỗi epoch.

- Đánh giá trên tập kiểm thử: Sau khi huấn luyện xong, mô hình cuối cùng được đánh giá trên tập kiểm thử để đo lường hiệu suất tổng quát.

4. Kết quả

Đường cong học tập (Learning Curve)

Đồ thị dưới đây biểu diễn sự thay đổi của giá trị mất mát (Loss) và độ chính xác (Accuracy) trên tập huấn luyện và tập xác thực qua các epoch.

- Loss Curve: Cho thấy giá trị mất mát giảm dần trên cả hai tập dữ liệu, chứng tỏ mô hình đang học tốt.
- Accuracy Curve: Cho thấy độ chính xác tăng dần, cho thấy khả năng phân loại của mô hình ngày càng được cải thiện.

Đồ thị đường cong học tập thể hiện loss và accuracy qua các epoch.

Ma trận nhầm lẫn (Confusion Matrix)

Ma trận nhầm lẫn là một công cụ trực quan để đánh giá hiệu suất của mô hình trên từng lớp. Các giá trị trên đường chéo chính thể hiện số lượng dự đoán đúng cho mỗi lớp.

Ma trận nhầm lẫn cho thấy hiệu suất phân loại trên 10 lớp của tập kiểm thử.

Từ ma trận, ta có thể thấy mô hình hoạt động khá tốt trên hầu hết các lớp. Một số nhầm lẫn xảy ra giữa các lớp có đặc điểm hình ảnh tương tự nhau, ví dụ như 'cat' và 'dog', hoặc 'truck' và 'automobile'.

5. Mã nguồn Python

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, random_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Kiểm tra xem có GPU không để sử dụng
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Using device: {device}')

# 1. Tải và chuẩn bị dữ liệu
# Định nghĩa các phép biến đổi cho dữ liệu
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Tải bộ dữ liệu CIFAR-10
train_dataset_full = torchvision.datasets.CIFAR10(root='./data', train=True,
                                                  download=True, transform=transform)
test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                            download=True, transform=transform)

# Chia tập huấn luyện thành tập huấn luyện và tập xác thực
train_size = int(0.8 * len(train_dataset_full))
val_size = len(train_dataset_full) - train_size
train_dataset, val_dataset = random_split(train_dataset_full, [train_size, val_size])

# Tạo các DataLoaders
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

# Các lớp của CIFAR-10
classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')

```

```python
# 2. Xây dựng mô hình CNN
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # Lớp tích chập 1
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        # Lớp tích chập 2
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        # Lớp tích chập 3
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Lớp kết nối đầy đủ
        # Kích thước ảnh sau 3 lớp pooling: 32 -> 16 -> 8 -> 4. Vậy là 4x4
        # Số kênh output của lớp conv3 là 128.
        self.fc1 = nn.Linear(128 * 4 * 4, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool1(self.relu(self.conv1(x)))
        x = self.pool2(self.relu(self.conv2(x)))
        x = self.pool3(self.relu(self.conv3(x)))
        # Làm phẳng tensor
        x = x.view(-1, 128 * 4 * 4)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = CNN().to(device)
print(model)

# 3. Định nghĩa hàm mất mát và trình tối ưu hóa
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 4. Huấn luyện mô hình
num_epochs = 20
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

for epoch in range(num_epochs):
    # Training
    model.train()
    running_loss = 0.0
    correct_train = 0
    total_train = 0
    for i, (images, labels) in enumerate(train_loader):
```

```
    for i, (images, labels) in enumerate(train_loader):
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    train_loss = running_loss / len(train_loader)
    train_acc = 100 * correct_train / total_train
    train_losses.append(train_loss)
    train_accuracies.append(train_acc)

    # Validation
    model.eval()
    running_loss = 0.0
    correct_val = 0
    total_val = 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total_val += labels.size(0)
            correct_val += (predicted == labels).sum().item()

    val_loss = running_loss / len(val_loader)
    val_acc = 100 * correct_val / total_val
    val_losses.append(val_loss)
    val_accuracies.append(val_acc)

    print(f'Epoch [{epoch+1}/{num_epochs}], '
          f'Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%, '
          f'Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%')

```

```python
# 5. Vẽ đường cong học tập
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(1, num_epochs + 1), train_losses, label='Train Loss')
plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Curve')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(1, num_epochs + 1), train_accuracies, label='Train Accuracy')
plt.plot(range(1, num_epochs + 1), val_accuracies, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
plt.title('Accuracy Curve')
plt.legend()
plt.tight_layout()
plt.show()

# 6. Đánh giá trên tập test và vẽ ma trận nhầm lẫn
model.eval()
all_labels = []
all_predictions = []
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        all_labels.extend(labels.cpu().numpy())
        all_predictions.extend(predicted.cpu().numpy())

print(f'Accuracy of the network on the 10000 test images: {100 * correct / total:.2f} %')

# Vẽ ma trận nhầm lẫn
cm = confusion_matrix(all_labels, all_predictions)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

I. Thiết lập Môi trường và Thư viện

Để chạy được đoạn mã này, bạn cần cài đặt Python và các thư viện khoa học dữ liệu liên quan.

Bước 1: Cài đặt Python Đảm bảo bạn đã cài đặt Python (phiên bản 3.8 trở lên được khuyến nghị) trên máy tính của mình.

Bước 2: Tạo Môi trường ảo (Khuyến nghị) Tạo một môi trường ảo sẽ giúp bạn quản lý các thư viện cho dự án này một cách độc lập, tránh xung đột với các dự án khác.

Bash

Tạo một môi trường ảo tên là "myenv"

python -m venv myenv

Kích hoạt môi trường trên Windows

myenv\Scripts\activate

Kích hoạt môi trường trên MacOS/Linux

source myenv/bin/activate

Bước 3: Cài đặt các thư viện cần thiết Sau khi kích hoạt môi trường ảo, bạn chỉ cần chạy một lệnh duy nhất để cài đặt tất cả các thư viện được sử dụng trong mã:

Bash

pip install torch torchvision matplotlib scikit-learn seaborn numpy

Bước 4: Thiết lập GPU (Tùy chọn nhưng quan trọng) Đoạn mã có dòng device = torch.device('cuda' if torch.cuda.is_available() else 'cpu') để tự động sử dụng GPU của NVIDIA (nếu có) nhằm tăng tốc độ huấn luyện một cách đáng kể.

- Để sử dụng GPU, bạn cần cài đặt **NVIDIA CUDA Toolkit** tương thích với driver card đồ họa của bạn.
- Sau đó, truy cập trang chủ của PyTorch (https://pytorch.org/) để lấy lệnh cài đặt PyTorch phiên bản hỗ trợ CUDA.

II. Giải thích chi tiết các thư viện

Mỗi thư viện đóng một vai trò chuyên biệt trong dự án này:

- **PyTorch (torch, torch.nn, torch.optim)**:

- Đây là **bộ não** của toàn bộ dự án.
- torch: Cung cấp cấu trúc dữ liệu cốt lõi là **Tensor**, một mảng đa chiều mạnh mẽ có khả năng tính toán trên GPU.
- torch.nn: Là module để xây dựng mạng nơ-ron. Bạn sử dụng nó để định nghĩa các lớp như lớp tích chập (nn.Conv2d), lớp gộp (nn.MaxPool2d), lớp kết nối đầy đủ (nn.Linear), và các hàm kích hoạt (nn.ReLU).
- torch.optim: Chứa các thuật toán tối ưu hóa như Adam hay SGD, dùng để cập nhật trọng số của mạng nơ-ron trong quá trình huấn luyện.

- **Torchvision (torchvision, torchvision.transforms)**:
    - Đây là **con mắt** và **người xử lý dữ liệu** của PyTorch, chuyên dùng cho các tác vụ thị giác máy tính.
    - torchvision.datasets: Cung cấp các bộ dữ liệu phổ biến như CIFAR-10, giúp bạn tải về chỉ bằng một dòng lệnh.
    - torchvision.transforms: Chứa các công cụ để tiền xử lý hình ảnh (ví dụ: chuyển ảnh sang Tensor, chuẩn hóa giá trị pixel).

- **Matplotlib (matplotlib.pyplot)**:
    - Đây là **họa sĩ** cơ bản, thư viện tiêu chuẩn để vẽ các loại biểu đồ trong Python. Trong mã này, nó được dùng để vẽ **đường cong học tập** (biểu đồ đường thể hiện sự thay đổi của loss và accuracy qua các epoch).

- **Seaborn (sns)**:
    - Đây là một **họa sĩ chuyên nghiệp hơn**, được xây dựng dựa trên Matplotlib để tạo ra các biểu đồ thống kê đẹp mắt và hấp dẫn hơn. Nó được dùng để vẽ **ma trận nhầm lẫn** (confusion_matrix) dưới dạng một bản đồ nhiệt (heatmap), giúp trực quan hóa hiệu suất của mô hình.

- **Scikit-learn (sklearn)**:
    - Là một **nhà thống kê** mạnh mẽ. Mặc dù nó có thể làm được nhiều việc, trong mã này chúng ta chỉ mượn một hàm duy nhất

là confusion_matrix để tính toán ma trận nhầm lẫn từ nhãn thật và nhãn dự đoán.

- **NumPy (np)**:
    - Là **nhà toán học** nền tảng cho khoa học dữ liệu trong Python. PyTorch và NumPy có thể chuyển đổi dữ liệu qua lại rất hiệu quả. Trong mã, nó được dùng ở bước cuối (.cpu().numpy()) để chuyển Tensor của PyTorch về mảng của NumPy, định dạng mà Scikit-learn và Matplotlib có thể hiểu được.

III. Chuẩn bị dữ liệu CIFAR-10

1. Định nghĩa các phép biến đổi (transforms.Compose) Mô hình nơ-ron không thể xử lý ảnh thô. Chúng ta cần chuyển đổi chúng về định dạng phù hợp. transforms.Compose cho phép ta kết hợp nhiều bước biến đổi lại với nhau.

- transforms.ToTensor(): Đây là phép biến đổi quan trọng nhất. Nó làm hai việc:
    1. Chuyển đổi hình ảnh (thường ở định dạng PIL Image) thành **PyTorch Tensor**.
    2. Chuẩn hóa giá trị của mỗi pixel từ dải [0, 255] về dải [0.0, 1.0].
- transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)):
    1. Sau khi đã về dải [0.0, 1.0], phép biến đổi này sẽ "chuẩn hóa" dữ liệu bằng cách trừ đi mean (0.5) và chia cho standard deviation (0.5) cho mỗi kênh màu (Đỏ, Lục, Lam).
    2. Kết quả là các giá trị pixel sẽ được đưa về dải [-1.0, 1.0]. Việc này giúp cho mạng nơ-ron hội tụ nhanh hơn và huấn luyện ổn định hơn.

2. Tải dữ liệu (torchvision.datasets.CIFAR10)

- Dòng lệnh này tự động tải bộ dữ liệu CIFAR-10 về thư mục ./data nếu nó chưa tồn tại (download=True).
- train=True để tải về 50,000 ảnh huấn luyện và train=False để tải về 10,000 ảnh kiểm thử.

- transform=transform áp dụng các phép biến đổi đã định nghĩa ở trên cho mỗi ảnh khi nó được tải.

3. Chia tập Huấn luyện và Xác thực (random_split)

- Chúng ta không dùng toàn bộ 50,000 ảnh để huấn luyện. Thay vào đó, chúng ta tách ra 20% (10,000 ảnh) để làm **tập xác thực (validation set)**.
- random_split là hàm của PyTorch giúp chia một dataset thành các phần nhỏ hơn một cách ngẫu nhiên và không trùng lặp. Tập xác thực này cực kỳ quan trọng để theo dõi hiện tượng **overfitting** trong khi huấn luyện.

2. Phân chia dữ liệu và tạo DataLoader

- **Phân chia dữ liệu (random_split)**
    - **Tại sao cần phân chia?** Chúng ta không bao giờ dùng toàn bộ dữ liệu để huấn luyện. Chúng ta cần một phần dữ liệu riêng gọi là **tập xác thực (validation set)** để kiểm tra xem mô hình có bị **quá khớp (overfitting)** hay không. Quá khớp là hiện tượng mô hình học thuộc lòng dữ liệu huấn luyện nhưng lại hoạt động kém trên dữ liệu mới.
    - **Cách thực hiện:** Đoạn mã lấy bộ dữ liệu huấn luyện gốc (train_dataset_full gồm 50,000 ảnh) và chia nó thành 2 phần theo tỉ lệ 80-20:
        - **train_dataset**: 40,000 ảnh (80%) để huấn luyện.
        - **val_dataset**: 10,000 ảnh (20%) để xác thực.
- **Tạo DataLoader**
    - **Mục đích:** DataLoader là một công cụ cực kỳ quan trọng giúp nạp dữ liệu vào mô hình một cách hiệu quả. Thay vì đưa toàn bộ 40,000 ảnh vào xử lý cùng lúc (sẽ làm tràn bộ nhớ), DataLoader sẽ cung cấp dữ liệu theo từng **lô nhỏ (mini-batch)**.
    - batch_size=64: Mỗi lần, DataLoader sẽ lấy ra 64 ảnh để đưa vào mô hình.
    - shuffle=True (cho train_loader): Ở mỗi epoch (vòng lặp huấn luyện), dữ liệu trong tập huấn luyện sẽ được xáo trộn. Điều này đảm bảo mô hình không học theo thứ tự của dữ liệu và giúp tăng tính tổng quát.

- shuffle=False (cho val_loader và test_loader): Khi đánh giá, chúng ta không cần xáo trộn để kết quả được nhất quán và có thể so sánh được.

IV: Xây dựng mô hình mạng Nơ-ron Tích Chập

1. Định nghĩa Kiến trúc Mô hình (class CNN)

a.__init__(self): Đây là nơi bạn "khai báo" tất cả các khối xây dựng (các tầng) của mạng. Giống như bạn mua các viên gạch Lego về và xếp ra đó.

- self.conv1, self.conv2, self.conv3: 3 tầng tích chập.

- self.pool1, self.pool2, self.pool3: 3 tầng gộp (max pooling).

- self.fc1, self.fc2: 2 tầng kết nối đầy đủ (fully connected) để phân loại.

b. forward(self, x): Đây là nơi bạn "lắp ráp" các viên gạch đã khai báo ở trên. Nó định nghĩa dòng chảy của dữ liệu (x) đi qua mạng:

- Dữ liệu x đi vào conv1, qua hàm kích hoạt ReLU, rồi vào pool1.

- Kết quả tiếp tục đi vào conv2 -> ReLU -> pool2.

- Tiếp tục vào conv3 -> ReLU -> pool3.

- Sau 3 tầng tích chập, dữ liệu là một khối 3D. x.view(-1, 128 * 4 * 4) sẽ "làm phẳng" nó thành một vector 1D.

- Vector này đi qua tầng fc1, qua ReLU, và cuối cùng là fc2 để cho ra 10 giá trị đầu ra (tương ứng với 10 lớp của CIFAR-10).

---

c. Giải thích Tầng Tích chập (nn.Conv2d)

Tầng tích chập là trái tim của Mạng Nơ-ron Tích chập (CNN). Nó hoạt động như một bộ lọc phát hiện đặc trưng.

- **Ý tưởng: Nó dùng một cửa sổ nhỏ gọi là kernel (bộ lọc) trượt trên toàn bộ ảnh đầu vào. Tại mỗi vị trí, nó thực hiện một phép nhân ma trận để tạo ra**

một giá trị duy nhất trong một bản đồ mới gọi là feature map (bản đồ đặc trưng).

- **Ví dụ: self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)**
    - **in_channels=3: Tầng này nhận đầu vào là một ảnh có 3 kênh (ảnh màu RGB).**
    - **out_channels=32: Tầng này sẽ sử dụng 32 bộ lọc khác nhau. Mỗi bộ lọc sẽ học cách phát hiện một đặc trưng khác nhau (ví dụ: một bộ lọc phát hiện cạnh dọc, một bộ lọc phát hiện góc cong, một bộ lọc phát hiện màu xanh lá...). Kết quả là nó sẽ tạo ra 32 bản đồ đặc trưng.**
    - **kernel_size=3: Mỗi bộ lọc có kích thước là 3x3 pixel.**
    - **padding=1: Thêm một đường viền dày 1 pixel xung quanh ảnh đầu vào. Việc này giúp cho kích thước của ảnh không bị giảm sau phép tích chập (khi kernel_size=3 và stride=1).**

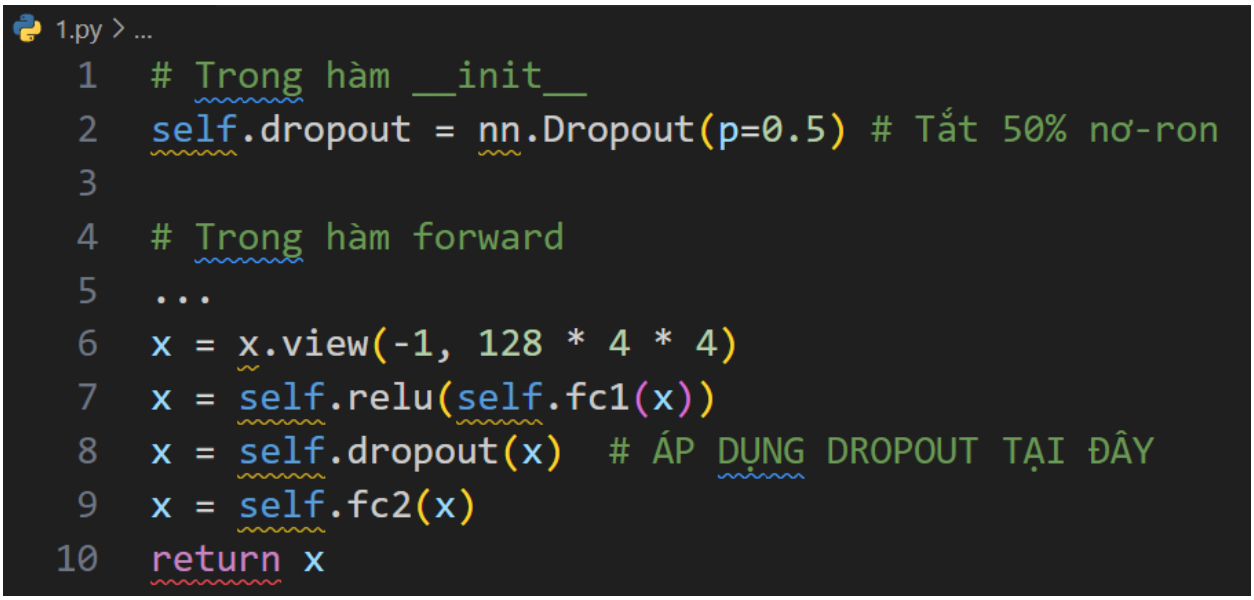
---

5. Vai trò của Dropout

Lưu ý quan trọng: Đoạn mã bạn cung cấp không sử dụng Dropout. Tuy nhiên, đây là một kỹ thuật rất phổ biến và quan trọng.

- **Mục đích: Dropout là một kỹ thuật chống quá khớp (regularization) cực kỳ hiệu quả.**
- **Cách hoạt động: Trong quá trình huấn luyện, ở mỗi bước, Dropout sẽ tắt ngẫu nhiên một tỷ lệ các nơ-ron trong một tầng nào đó (ví dụ 50%).**
- **Tác dụng:**
    - **Nó buộc mạng không được phụ thuộc quá nhiều vào bất kỳ một nơ-ron hay một đặc trưng nào.**
    - **Nó giống như việc mỗi lần làm bài tập nhóm, một vài thành viên vắng mặt ngẫu nhiên, buộc các thành viên còn lại phải nỗ lực và linh hoạt hơn.**
    - **Kết quả là mạng học được các đặc trưng mạnh mẽ và có tính tổng quát cao hơn.**

- **Cách thêm vào mã: Bạn có thể thêm một tầng Dropout vào giữa hai tầng kết nối đầy đủ:**



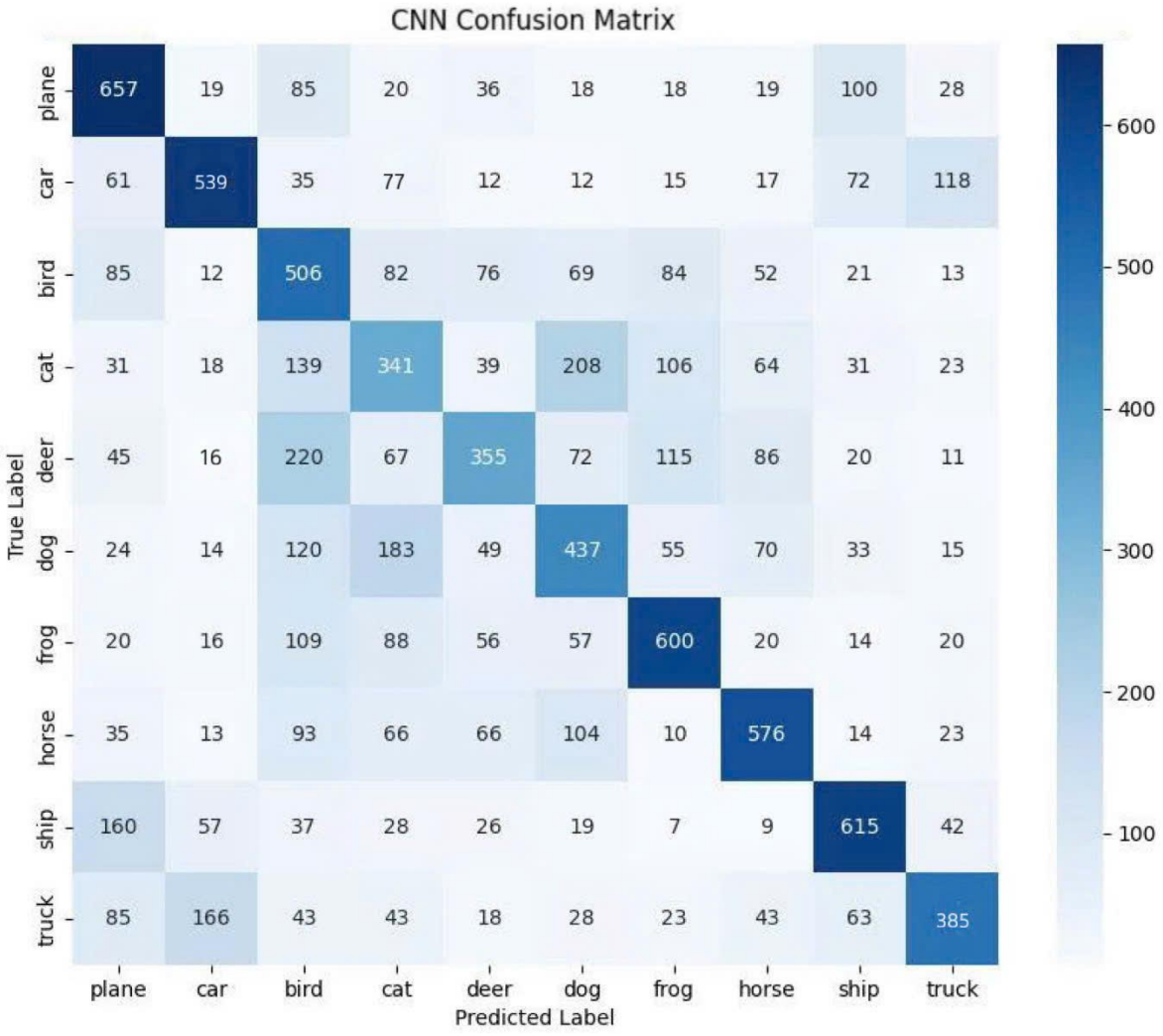
6. Huấn luyện và Đánh giá Mô hình

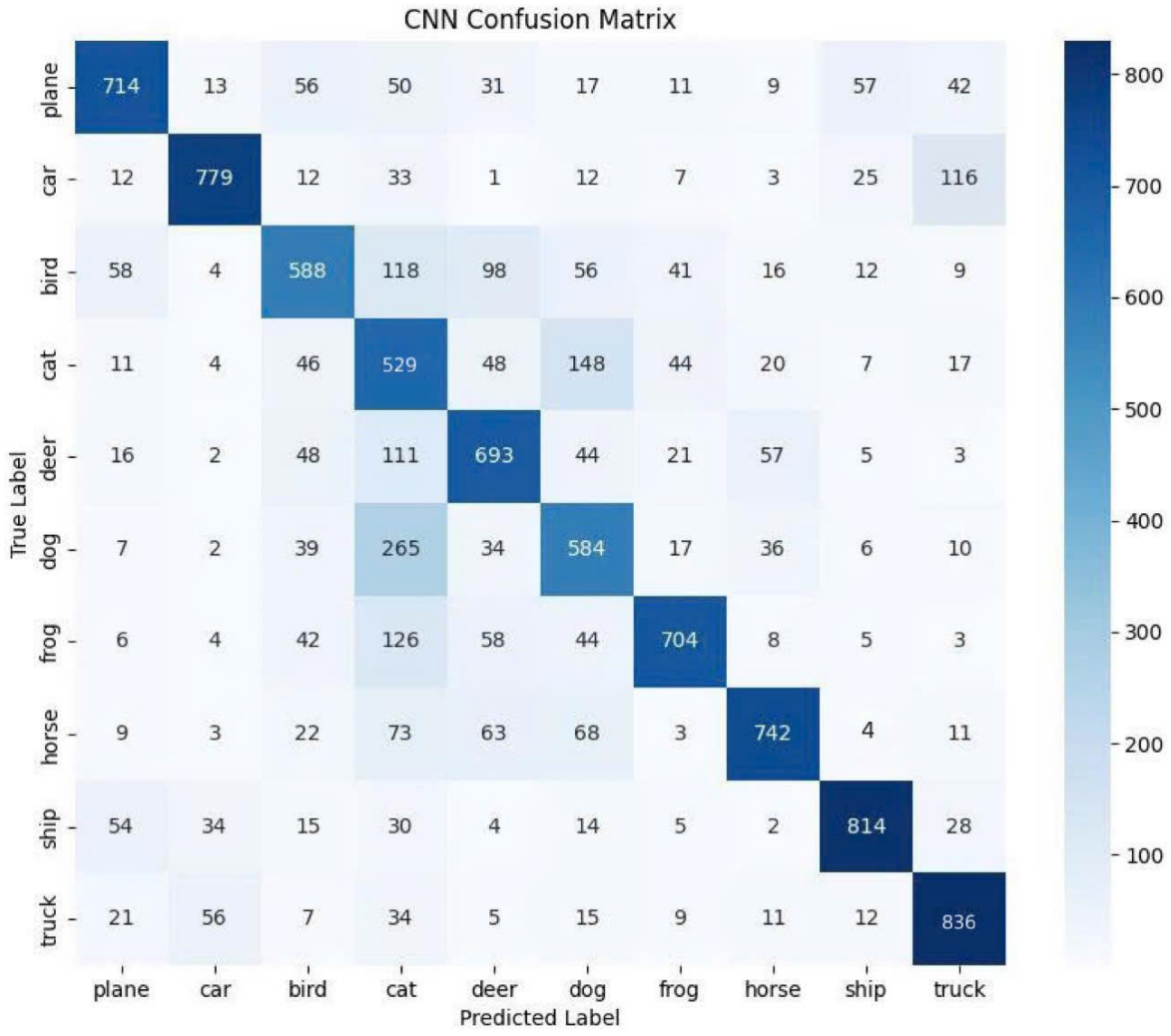
Đây là vòng lặp chính nơi mô hình học và được kiểm tra.

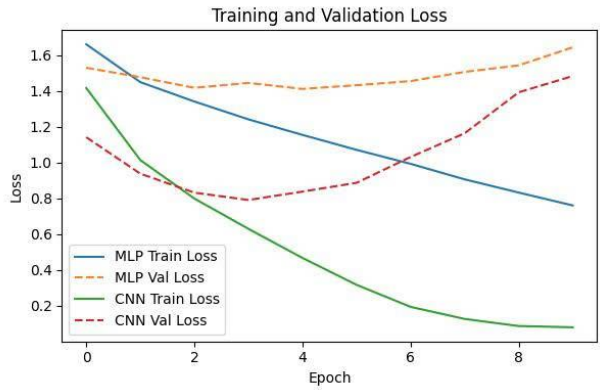
- for epoch in range(num_epochs): Vòng lặp lớn bên ngoài, lặp lại toàn bộ quá trình huấn luyện trong num_epochs lần.
- **Pha Huấn luyện (Training Phase):**
    - model.train(): Bật **chế độ huấn luyện**. Chế độ này sẽ kích hoạt các cơ chế như Dropout (nếu có) và cho phép tính toán gradient.
    - optimizer.zero_grad(): Xóa sạch các gradient từ bước lặp trước đó. Đây là việc bắt buộc phải làm đầu tiên.
    - outputs = model(images): Lan truyền xuôi (forward pass).
    - loss = criterion(outputs, labels): Tính toán lỗi.
    - loss.backward(): Lan truyền ngược (backpropagation) để tính toán gradient.
    - optimizer.step(): Cập nhật trọng số của mô hình dựa trên gradient vừa tính.

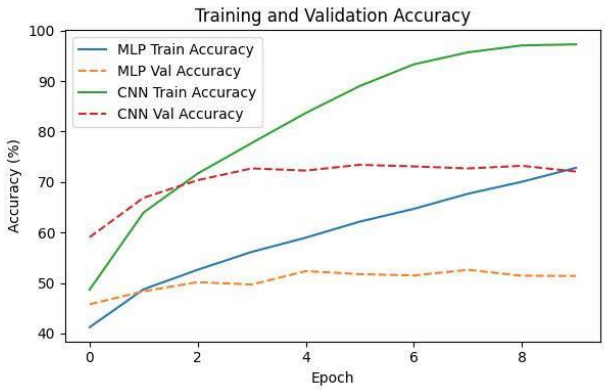
- **Pha Đánh giá (Validation Phase):**
    - model.eval(): Bật **chế độ đánh giá**. Chế độ này sẽ tắt các cơ chế như Dropout và đảm bảo mô hình cho ra kết quả nhất quán.
    - with torch.no_grad(): Tắt việc tính toán gradient trong khối lệnh này. Vì chúng ta chỉ đang đánh giá, không cần tính gradient, việc này giúp tiết kiệm bộ nhớ và tăng tốc độ xử lý.
    - Phần còn lại của vòng lặp chỉ đơn giản là thực hiện lan truyền xuôi và tính toán loss/accuracy trên tập xác thực để theo dõi hiệu suất của mô hình.

Kết quả:




CNN Confusion Matrix
True Label
Predicted Label
plane
car
bird
cat
deer
dog
frog
horse
ship
truck
714 13 56 50 31 17 11 9 57 42
12 779 12 33 1 12 7 3 25 116
58 4 588 118 98 56 41 16 12 9
11 4 46 529 48 148 44 20 7 17
16 2 48 111 693 44 21 57 5 3
7 2 39 265 34 584 17 36 6 10
6 4 42 126 58 44 704 8 5 3
9 3 22 73 63 68 3 742 4 11
54 34 15 30 4 14 5 2 814 28
21 56 7 34 5 15 9 11 12 836
800
700
600
500
400
300
200
100


Training and Validation Loss
Loss
Epoch
MLP Train Loss
MLP Val Loss
CNN Train Loss
CNN Val Loss


Training and Validation Accuracy
Accuracy (%)
Epoch
MLP Train Accuracy
MLP Val Accuracy
CNN Train Accuracy
CNN Val Accuracy